Ravens Project

Manqing Mao
mmao33@gatech.edu

1. Approach Overview

Q1: Provide a narrative of how you approached the project.

Goal: In this project, I am going to develop an agent to solve a set of problems inspired by the Raven's Progressive Matrices test of intelligence by using visual approach. The agent developed based on training dataset has to be general enough to also solve the problems in test datasets. Compared with Problem Set B and C, Problem Set D and E do not have verbal description so that visual method has to be developed.

Approach: In this project, I transferred from verbal representation to visual representation to read in all figures. For each problem, including 7 figures for problem description and 8 figures for candidate solutions. After reading in the 7 figures about problem description, the agent solves the problems from different problem set (namely Problem Set D and Problem Set E) by using different solver functions. For each problem set, I developed three functions so that problems in a certain set can be solved by using different functions. These six functions will be described later.

Q2: How did your approach change over time and what modifications did you make?

Four versions of agent: I developed three versions of agent over time, namely, Agent_v1, Agent_v2, Agent_v3, respectively.

> **Agent_v1:** Agent_v1 develop two functions for Problem Set D, namely *solution_AtoC* and *solution_BDandAE*; as well as two functions for Problem Set E, namely *solution_union* and *solution_intersection*.
>
> - ***solution_AtoC:*** This function defines the relationship, which only involves the delta from A to C. In order to determine if the problem follows this rule, we impose the constraint to check if delta(A, C) = delta(G, ***x***).

if we find delta(A, C) = delta(G, ***x***):

then we guess, # = ***x***.

Problem can be solved: D-01, D-04, D-05, D-06.

- ***solution_BDandAE:*** This function defines the relationship, which uses several patterns to form the answer. In order to determine if the problem follows this rule, we impose the constraint to check if A = union(intersection(F, H), intersection(A, E)).

  if we find A = union(intersection(F, H), intersection(A, E))

  then we guess, # = union(intersection(B, D), intersection(A, E)).

  Problem can be solved: D-02, D-07, D-10, D-11.

- ***solution_union:*** This function defines the union relationship. In order to determine if the problem follows this rule, we impose the constraint to check if C = union(A, B) and F = union(D, E).

  if we find C = union(A, B) **and** F = union(D, E):

  then we guess, # = union(G, H).

  Problem can be solved: E-01, E-02, E-03, E-09.

- ***solution_intersection:*** This function defines the intersection relationship. In order to determine if the problem follows this rule, we impose the constraint to check if C = intersection(A, B) and F = intersection (D, E).

  if we find C = intersection(A, B) **and** F = intersection(D, E):

  then we guess, # = intersection(G, H).

  Problem can be solved: E-10, E-11.

	Agent_v1	Agent_v2	Agent_v3
Functions for PS-D	solution_AtoC solution_BDandAE	Same as Agent_v1	… (Agent_v2) + **solution_remaining_pattern**
Functions for PS-E	solution_union solution_intersection	… (Agent_v1) + **solution_GtoH**	Same as Agent_v2
Problem Solved in PS-D	01, 04, 05, 06, 02, 07, 10, 11	01, 02, 04, 05, 06, 07, 10, 11	01, 02, 04, 05, 06, 07, 10, 11, **03, 09**
Problem Solved in PS-E	01, 02, 03, 09, 10, 11	01, 02, 03, 09, 10, 11 **05, 06, 07, 08**	01, 02, 03, 05, 06, 07, 08, 09, 10, 11

Agent_v2: Other than all relationships include in Agent_v1, Agent_v2 develop one more function for Problem Set E, namely *solution_GtoH.*

- ***solution_GtoH:*** This function defines the relationship, which subtract B from A to get C. In order to determine if the problem follows this rule, we impose the constraint to check if C = substract(A, B).

  if we find C = substract(A, B)

  then we guess, # = substract(G, H).

  Problem can be solved: E-05, E-06, E-07, E-08.

Agent_v3: Other than all relationships include in Agent_v2, Agent_v3 develop one more function for Problem Set D, namely *solution_remaining_pattern.*

- ***solution_remaining_pattern:*** This function defines the relationship, which needs to collect all patterns used by A, B and C; then C is actually formed by the patterns which not belong to A and B. In order to determine if the problem follows this rule, we impose the constraint to check if G = union(patterns'); H = union(patterns'').

  if we find G = union(patterns') **and** H = union(patterns'')

  then we guess, # = union(remaining patterns)

Problem can be solved: D-02, D-03, D-09, D-10, D-11.

2. Process in Final Agent – Agent_v3

Q1: How does the final agent work? What is its process of solving the problem? How does the final agent represent the information about the problem?

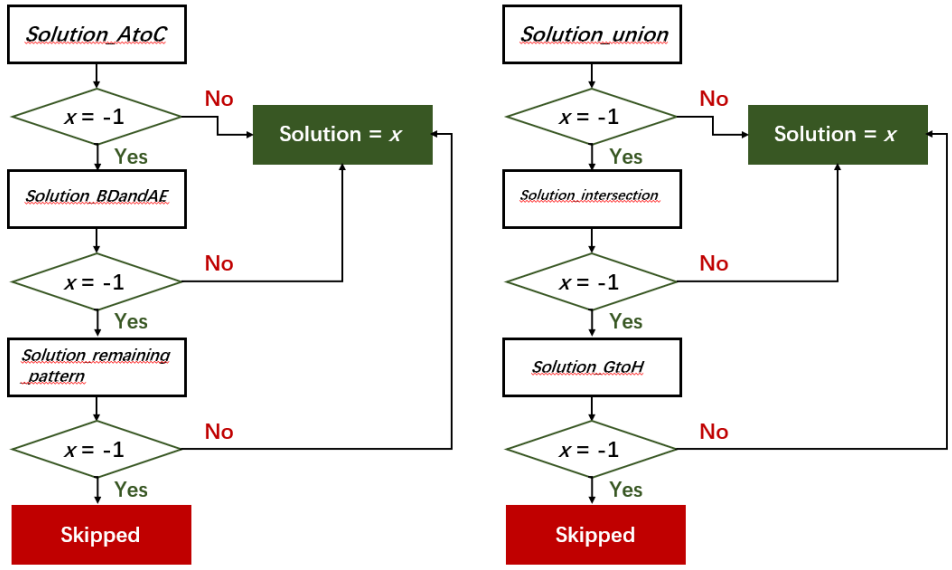
Figure 1 shows the procedure for Agent_v3 to solve the problem. In this figure, we can see that we actually separate two problem sets and solve each problem set by using different functions, as described in the Section 1.

(1) At beginning, the agent determines what the problem set is. If this is the Problem Set D, the agent will call *solver_problem_D;* If this is the Problem Set E, the agent will call *solver_problem_E.*

(2) In *solver_problem_D*, as shown in Fig. 1 (a), the agent tries to solve the problem by using the first function (***solution_AtoC***). If the problem is not solved, we feed this problem to the second function (***solution_BDandAE***); If the problem is not solved by the second function, we feed this problem to the third function (***solution_remaining_pattern***). If the problem cannot be solved by using these three functions, we skipped the question and return -1.

(3) In *solver_problem_E*, as shown in Fig. 1 (b), the agent tries to solve the problem by using the first function (***solution_union***). If the problem is not solved, we feed this problem to the second function (***solution_intersection***); If the problem is not solved by the second function, we feed this problem to the third function (***solution_GtoH***). If the problem cannot be solved by using these three functions, we skipped the question and return -1.

It is worth to mention that no matter what the order for these three functions is, agent always gives the same performance. This is because I properly impose the contraints for each funtion by tuning the threshold which determines the relationship. In this way, the function will **either** solve the corresponding question corretly **or** skips the question **instead of** giving the wrong answer. This is actually very important by using a such method.



(a) (b)

Figure 1. (a) Basic problem D solver and (b) Basic problem E solver in Agent_v3.

3. Performance in Final Agent – Agent_v3

Q1: How many problems does it answer correctly? How efficient is it? How general is it? Does its performance on the Basic and Test sets differ significantly, or are they about the same?

So far, final agent can successfully solve the Basic Problem Set D: 10 out of 12; Test Problem Set D: 8 out of 12; Basic Problem Set E: 10 out of 12; Test Problem Set E: 9 out of 12. It is pretty fast since I just coded about 260 lines to solve the basic questions. The agent solves problems by pre-defining the relationship in different functions and trying to find answer. It is also quite general since we can see the similar performance for basic set and testing set.

Performance	Agent_v1	Agent_v2	Agent_v3
Basic/Test PS-D	8/4	8/4	10/8
Basic/Test PS-E	6/6	10/9	10/9

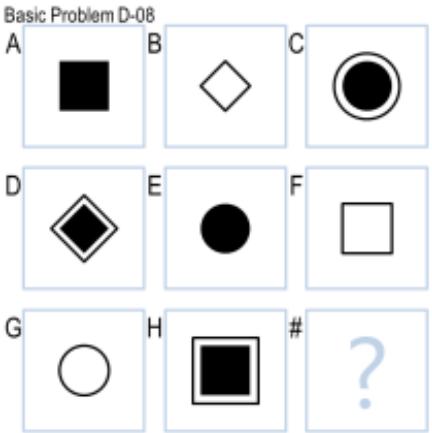
4. Limitation in Final Agent – Agent_v3

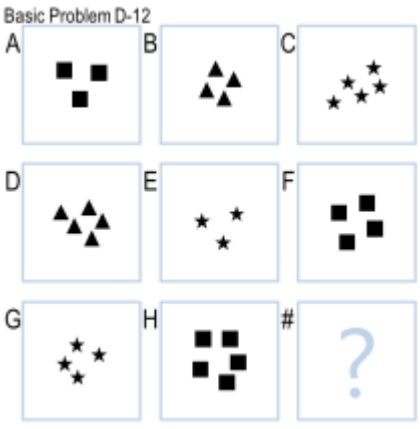
Q1: What are its limitations? What types of problems does it currently answer incorrectly?

Limitations: My final agent does not consider very complicated relationship except the above 3 relationships described in previous sections. Thus, it does not have good performance on Raven's dataset. (My final agent can only answer correctly about 5 out of 12 Raven's problem in Problem Set D and 7 out of 12 Raven's problem in Problem Set E.)

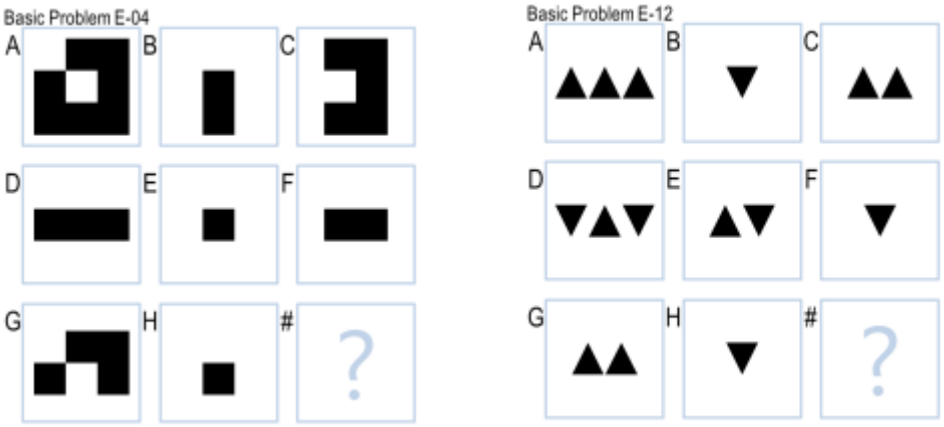
What Type of Problems it Fails: (1) Basic Problem Set D-08 and D-12, which involve either more than two simple relationships (D-08) or the number of objects (D-12). (2) Basic Problem Set E-04 and E-12, which also involve more than two simple relationships So far, my agent only solves problem by using three simple functions for each problem set.

How to Solve the Limitation in the Future Work: I have to add more functions as well as determine threshold properly to make sure each function will either solve the problem correctly or skips the question instead of giving the wrong answer. Otherwise, the function order matters. However, it is kind of tricky since the training dataset is always different from testing dataset, which is actually hidden for us. As more and more functions added, the overfitting issues will be bigger and bigger.





(a)



(b)

Figure 3. Failed basic problems for (a): D-08, D-12 and (b): E-04, E-12.